Materiale didattico di supporto alle lezioni del corso di

Linguaggio di programmazione C++

Corso di Laurea in Ingegneria Biomedica

Prof. Cinzia Bernardeschi
Dipartimento di Ingegneria dell'Informazione

Anno Accademico 2021-2022

2.1 Linguaggio di Programmazione C++ (I)

- Per definire sintassi e semantica di un linguaggio occorre utilizzare un altro linguaggio, ossia un *metalinguaggio*

- Metalinguaggio per la sintassi C++:
  - insieme di notazioni (non ambigue), che possono essere spiegate con poche parole del linguaggio naturale.

- Metalinguaggio per la semantica C++:
  - risulta assai complesso, per cui si ricorre direttamente al linguaggio naturale.

- Notazione utilizzata per la sintassi C++:
  - derivata dal classico formalismo di Backus e Naur (*BNF*, *Backus-Naur Form*).

2.1 Metalinguaggio per il C++ (I)

NOTAZIONE UTILIZZATA

basata sulla grammatica BNF; terminologia inglese;

rispetto alla sintassi *ufficiale*, regole semplificate, caratterizzate dal prefisso *basic*;

diversa organizzazione delle categorie sintattiche.

Regole

- una regola descrive una *categoria sintattica*, utilizzando altre categorie sintattiche, costrutti di metalinguaggio, simboli terminali

- le forme alternative possono stare su righe separate, oppure essere elencate dopo il simbolo del metalinguaggio <u>one of</u> .

Categorie sintattiche:

– scritte in *corsivo*.

Costrutti di metalinguaggio:

– scritti con <u>sottolineatura</u>.

Simboli terminali:

– scritti con caratteri normali.

2.1 Metalinguaggio per il C++ (III)

Elementi di una categoria sintattica:

- possono essere opzionali
  vengono contrassegnati con il suffisso |opt (simbolo di metalinguaggio)
- possono essere ripetuti più volte
  per far questo, vengono introdotte categorie sintattiche aggiuntive.

Categorie sintattiche aggiuntive

1. Sequenza di un qualunque genere di elementi:

some-element-seq
some-element
some-element some-element-seq

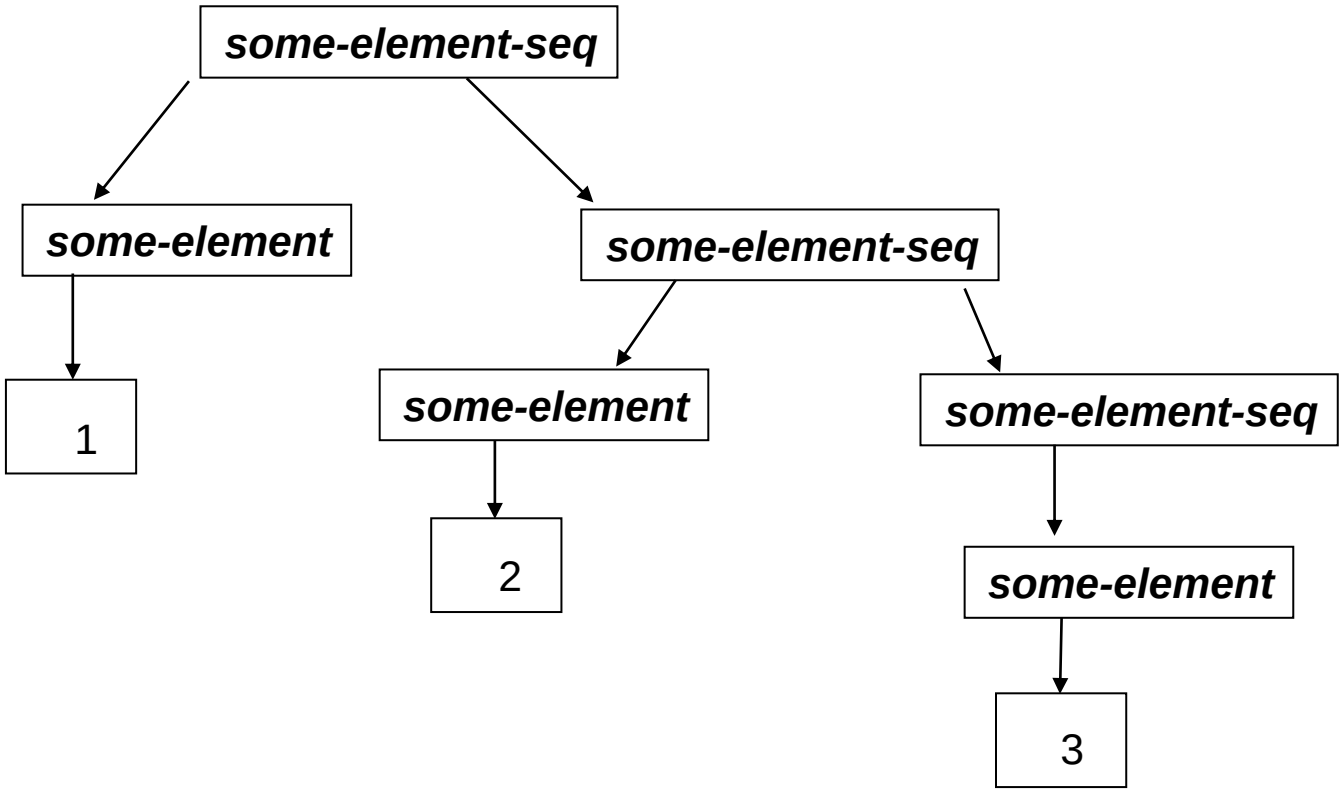
2. Lista di un qualunque genere di elementi (separati da virgola):

some-element-list
some-element
some-element , some-element-list

2.1 Metalinguaggio per il C++ (IV)

Albero di derivazione per la sequenza: 1 2 3

some-element
one of
0 1 2 3 4 5 6 7 8 9



2.2 Sintassi C++ (I)

Programma C++:

- costituito da sequenze di *parole* (*token*);
- le parole possono essere delimitate da *spazi bianchi* (*whitespace*).

Parole sono costituite dai seguenti caratteri:

token-character
digit
letter
special

digit
one of
0 1 2 3 4 5 6 7 8 9

letter
one of
_ a b ... z A B ... Z

special
one of
! % ^ ... /

2.2 Sintassi C++ (II)

Spazi bianchi:

- carattere *spazio*;
- caratteri *tabulazione* (orizzontale e verticale);
- caratteri *nuova riga* e *ritorno carrello*.

Commenti:

- sequenze di parole e spazi bianchi racchiuse fra i caratteri /* e */, oppure fra i caratteri // e la fine della riga;
- hanno lo scopo di documentare un programma;
- possono essere inseriti liberamente nel testo e non hanno alcun effetto sull'esecuzione del programma.

Spazi bianchi e commenti:

- costituiscono le *spaziature*.

2.2 Sintassi C++ (III)

Categorie sintattiche elementari (elementi lessicali):

- opportune sequenze di caratteri (token-character o whitespace);
- non possono includere spaziature (aggiuntive) fra un carattere e un altro.

Elementi lessicali:

- identificatori (*identifier*);
- parole chiave (*keyword*);
- espressioni letterali (*literal*);
- operatori (*operator*);
- separatori *(separator*).

2.2.1 Identificatori

Entità usate in un programma:

- devono possedere *nomi*;
- i nomi possono essere *identificatori*:

identifier
letter
letter identifier-char-seq
identifier-char
letter
digit

Il carattere di sottolineatura _ è una lettera.

- la doppia sottolineatura all'interno degli identificatori è sconsigliata, perché riservata alle implementazioni ed alle librerie.

Il C++ distingue fra maiuscole e minuscole (è *case sensitive*).
Esempi: ident _ident Ident

2.2.2 Parole Chiave e Espressioni Letterali

Nota:

- i termini *nome* e *identificatore* spesso vengono usati intercambiabilmente, ma è necessario distinguerli:

  un nome può essere un identificatore, oppure un identificatore con altri simboli aggiuntivi.

Parole chiave:

- simboli costituiti da parole inglesi (formate da sequenze di lettere), il cui significato è stabilito dal linguaggio:

keyword

one of

and ... while

Un identificatore non può essere uguale ad una parola chiave.

2.2.2 Parole Chiave e Espressioni Letterali

Espressioni letterali:

- chiamate semplicemente *letterali*;
- denotano valori costanti (costanti senza nome);

  - numeri interi (per es. 10);
  - numeri reali (per es. –12.5);
  - letterali carattere (per es. ‘a’);
  - letterali stringa (per es. “informatica”).

2.2.4 Operatori e separatori

Operatori:

- caratteri speciali e loro combinazioni;
- servono a denotare operazioni nel calcolo delle espressioni;
- esempi:
  - carattere +
  - carattere -
  - …

Separatori:

- simboli di interpunzione, che indicano il termine di una istruzione, separano elementi di liste, raggruppano istruzioni o espressioni, eccetera;
- esempi:
  - carattere ;
  - coppia di caratteri ( )
  - …

2.2.4 Proprietà degli operatori (I)

- **posizione rispetto ai suoi operandi (o argomenti):**
  - *prefisso*: se precede gli argomenti

    op arg

    dove *op* e' l'operatore e *arg* e' l'argomento

    Esempio: +5
  - *postfisso*: se segue gli argomenti

    arg op

    Esempio: x++ (operatore incremento)
  - *infisso*: in tutti gli altri casi;

    arg1 op arg2

    Esempio: 4 + 5
- **numero di argomenti (o arietà):**

  Esempio: *op arg* (arietà 1)

  arg1 op arg2 (arietà 2)

2.2.4 Proprietà degli operatori (II)

- ***precedenza (o priorità) nell'ordine di esecuzione:***
  - gli operatori con priorità più alta vengono eseguiti per primi;

    Esempio: *arg1 + arg2 * arg3*

    (operatore prodotto priorità maggiore dell'operatore somma)

- **associatività (ordine in cui vengono eseguiti operatori della stessa priorità):**
  - operatori associativi a sinistra: vengono eseguiti da sinistra a destra;

    Esempio: *arg1 + arg2 + arg3*

    ----- *(arg1 + arg2) + arg3*

  - operatori associativi a destra: vengono eseguiti da destra a sinistra.

    Esempio: *arg1 = arg2 = arg3*

    ----- *arg1 = (arg2 = arg3)*

3. Un semplice programma

Il più semplice programma C++

```
int main()
{ }
```

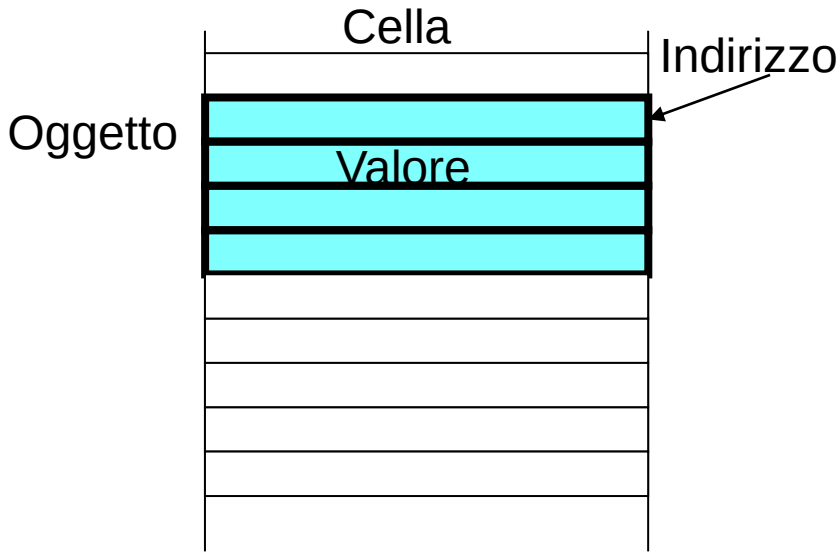
Un passo avanti

```
#include <cstdlib>  // direttive per il prepocessore
using namespace std;
        /*direttiva che indica al compilatore che tutti i nomiusati nel programma
          si riferiscono allo standard ANSI-C++      */

int main()              // dichiarazione della funzione main
{
    system("PAUSE");
    /* consente di bloccare l'esecuzione in attesa che l'utente  digiti un tasto*/

    return 0; /* restituisce 0 ovverro tutto OK!!!! */
}
```

```
Premere un tasto per continuare . . .
```

3.1 Oggetti (I)

Memoria: insieme di celle.
Cella: in genere dimensione di un byte (8 bit)



Oggetto: gruppo di celle consecutive che vengono considerate dal programmatore come un'unica cella informativa.

Attributi di un oggetto:

Indirizzo della prima cella

Valore (contenuto di tutte le celle)

3.1 Oggetti (II)

Oggetti costanti (costanti con nome) e oggetti variabili:

- l'indirizzo comunque non cambia;
- il valore non può o può subire modifiche, rispettivamente.

Programmatore:

- si riferisce a un oggetto mediante un nome (caso particolare di nome: identificatore).

Oggetto:

- ha un tipo.

Tipo di un oggetto:

- insieme di valori (detti elementi o costanti del tipo);
- insieme di operazioni definite sugli elementi (con risultato appartenente allo stesso tipo o ad un altro tipo).

Associare un tipo a un oggetto:

- permette di rilevare in maniera automatica valori che non siano compresi nell'insieme di definizione e operazioni non consentite.

3.2 Dichiarazioni e Definizioni

Costrutti che introducono nuove entità:

- dichiarazioni;
- definizioni.

Dichiarazioni:

- entità a cui il compilatore non associa locazioni di memoria o azioni eseguibili. Esempio: dichiarazioni di tipo.

Definizioni:

- entità a cui il compilatore associa locazioni di memoria o azioni eseguibili. Esempio: definizioni di variabili o di costanti (con nome).

Nomenclatura consentita in C++:

- spesso non è semplice né conveniente trattare separatamente dichiarazioni e definizioni;
- con *dichiarazione* si può intendere sia una *dichiarazione* vera e propria sia una *definizione* (le dichiarazioni comprendono le definizioni).

3.2 Tipi del C++

Tipi: Tipi fondamentali
Tipi derivati

Tipi fondamentali (chiamati anche *tipi aritmetici)*:

- tipi predefiniti;
- tipi enumerazione.

Tipi predefiniti:

- tipo intero (int) e tipo naturale (unsigned);
- tipo reale (double);
- tipo booleano (bool);
- tipo carattere (char).

Il tipo intero e il tipo reale sono detti tipi *numerici*.
Il tipo intero, booleano, carattere ed i tipi enumerati sono detti *tipi discreti*.

3.2 Tipi del C++

Tipi enumerazione:

- Esempio: tipo colore definito per enumerazione
  {rosso, giallo, verde};

Tipi derivati:

- si ottengono a partire dai tipi predefiniti;
- permettono di costruire strutture dati più complesse.
- Esempio: un riferimento ad un oggetto già esistente

3.3 Tipo intero (I)

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int i;
    int i1 = 7;
    int i2(7);
    int i3 = 0, i4, i5 = 6;
    i1 = -7;                    // i1 = -7 (cambiamento di segno)
    i2 = i1 + 3;                // i2 = -4 (somma)
    i2 = i1 - 1;                // i2 = -8 (sottrazione)
    i2 = i1 * 2;                // i2 = -14 (moltiplicazione)
    i4 = 1 / 2;                     // i4 = 0   (quoziente)
    i5 = 1 % 2;                 // i5 = 1   (resto)
    i3 = 1 / 2 * 2 + 1 % 2;     // i3 = 1   (a=(a/b)*b + a%b)
    cout << i3 << endl;
    return 0;
}
```

```
1
Premere un tasto per continuare . . .
```

3.3 Tipo intero (II)

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    // tipo short int
    short int s1 = 1;        // letterale int
    short s2 = 2;

    // tipo long int
    long int ln1 = 6543;     // letterale int
    long ln2 = 6543L;        // letterale long int (suffisso L)
    long ln3 = 6543l;        // letterale long int (suffisso l)

    // letterale int ottale, prefisso 0 (zero)
    int ott = 011;           // ott = 9 (letterale intero ottale)

    // letterale int esadecimale, prefisso 0x o 0X
    int esad1 = 0xF;         // esad1 = 15
    int esad2 = 0XF;         // esad2 = 15
```

3.3 Tipo intero (II)

```
  cout << ott << endl << esad1 << endl;
  cout << esad2 << endl;
  return 0;
}
```

```
9
15
15
Premere un tasto per continuare . . .
```

3.3 Tipo intero (III)

Definizione di un intero con il formalismo di Backus-Naur

basic-int-definition
int ***int-specifier-list*** **;**

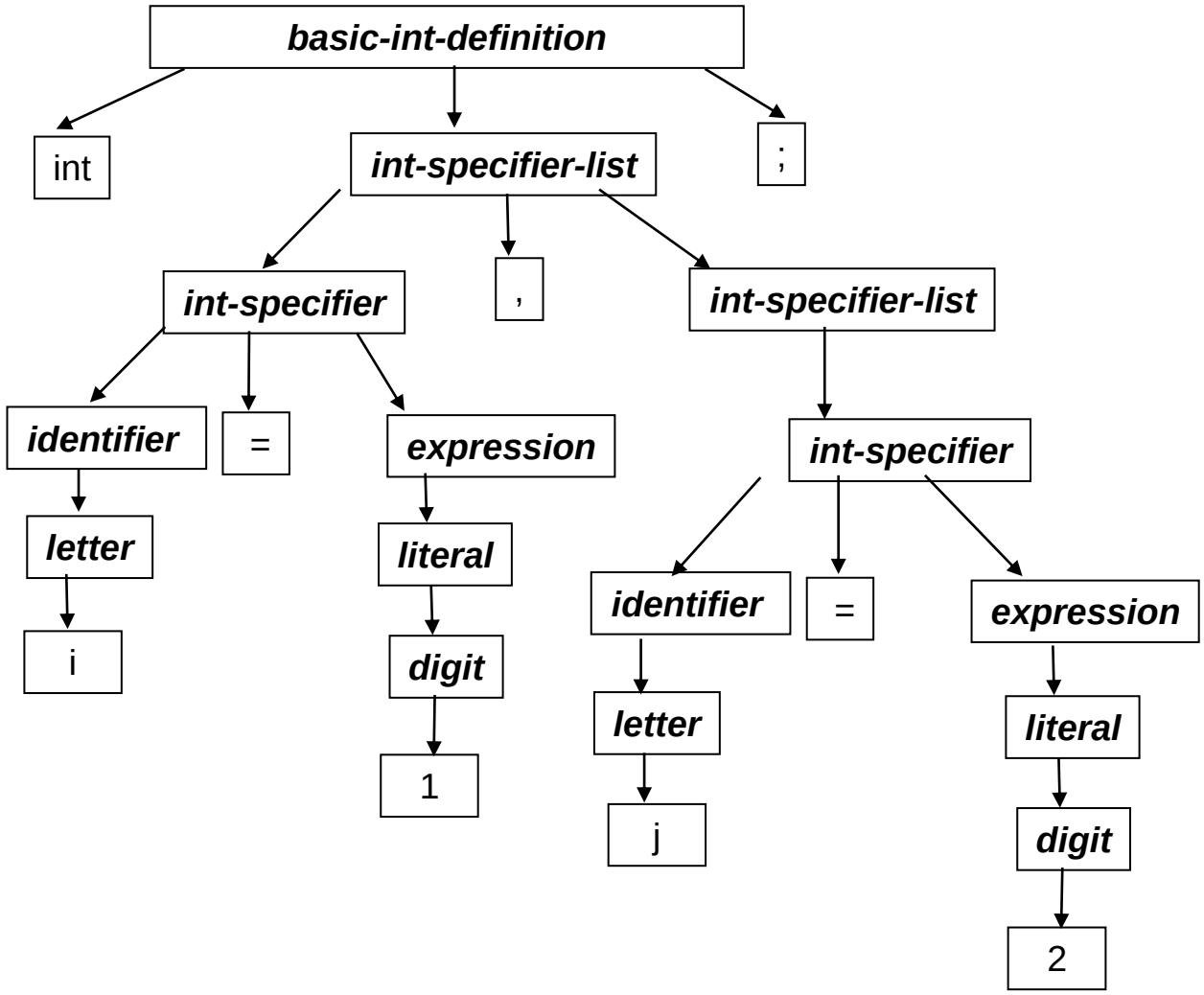
int-specifier-list
int-specifier
int-specifier**,** ***int-specifier-list***

int-specifier
identifier int-initializer**|opt**

int-initializer
= ***expression***
(***expression*** **)**

3.3 Tipo Intero (IV)

Albero di derivazione per la definizione: int i = 1, j = 2;



3.3.1 Tipo unsigned (I)

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{    // tipo unsigned int
     unsigned int u1 = 1U;
                                  // letterale unsigned, suffisso U
     unsigned u2 = 2u;  // letterale unsigned, suffisso u

     // tipo unsigned short int
     unsigned short int u3 = 3;
     unsigned short u4 = 4;

     // tipo unsigned long int
     unsigned long int u5 = 5555;
     unsigned long u6 = 6666UL;
     unsigned long u7 = 7777LU;
                    // letterale unsigned long, suffisso UL (ul)

     unsigned short int u8 = -0X0001;          // Warning
```

3.3.1 Tipo unsigned (I)

```
    cout << u1 << '\t' << u2 << endl;
    cout << u3 << '\t' << u4 << endl;
    cout << u5 << '\t' << u6 << '\t' << u7 << endl;
    cout << u8 << endl;
    system("PAUSE");
    return 0;
}
```

```
1       2
3       4
5555    6666    7777
65535
Premere un tasto per continuare . . .
```

3.3.1 Tipo unsigned (II)

Osservazioni:

- se N è il numero di bit impiegati per rappresentare gli interi, i valori vanno da 0 a 2**N - 1;

- Il tipo **unsigned** è utilizzato principalmente per operazioni a basso livello:
  - il contenuto di alcune celle di memoria non è visto come un valore numerico, ma come una configurazione di bit.

Operatori bit a bit:

| | OR bit a bit
| & | AND bit a bit
| ^ | OR esclusivo bit a bit
| ~ | complemento bit a bit
| << | traslazione a sinistra
| >> | traslazione a destra

3.3.1 Tipo unsigned (II)

a	b	|	&	^	~
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

3.3.1 Tipo unsigned (III)

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    unsigned short a = 0xFFF9;
                        // 1111 1111 1111 1001 (65529)
    unsigned short b = ~a;
                        // 0000 0000 0000 0110 (6)
    unsigned short c = 0x0013;
                        // 0000 0000 0001 0011 (19)

    unsigned short c1, c2, c3;
    c1 = b | c;         // 0000 0000 0001 0111 (23)
    c2 = b & c;         // 0000 0000 0000 0010 (2)
    c3 = b ^ c;         // 0000 0000 0001 0101 (21)

    unsigned short b1, b2;
    b1 = b << 2;        // 0000 0000 0001 1000 (24)
    b2 = b >> 1;        // 0000 0000 0000 0011 (3)
```

3.3.1 Tipo unsigned (III)

```
    cout << a << '\t' << b << '\t' << c << endl;
    cout << c1 << '\t' << c2 << '\t' << c3 << endl;
    cout << b1 << '\t' << b2 << endl;
    system("PAUSE");
    return 0;
}
```

```
65529   6       19
23      2       21
24      3
Premere un tasto per continuare . . .. . .
```

3.4 Tipo reale (I)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
        // tipo double
    double d1 = 3.3;
    double d2 = -12.14e-3, d3 = 1.51;

        // tipo float
    float f = -2.2;
    float g = f - 12.12F;
            // letterale float, suffisso F (f)

    long double h = +0.1;
    long double k = 1.23e+12L;
            // letterale long double, suffisso L (l)
```

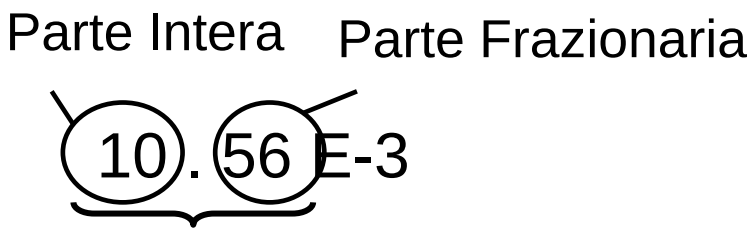
3.4 Tipo reale (I)

```
    cout << d1 << '\t' << d2 << '\t' << d3 << endl;
    cout << f << '\t' << g << endl;
    cout << h << '\t' << k << endl;

    system("PAUSE");
    return 0;
}
```

```
3.3     -0.01214        1.51
-2.2    -14.32
0.1     1.23e+012
Premere un tasto per continuare . . .
```

3.4 Tipo reale (II)

Letterale reale (forma estesa):



- **la parte intera o la parte frazionaria, se valgono zero, possono essere omesse.**

Le operazioni sugli interi e sui reali si indicano con gli stessi simboli (*sovrapposizione* o *overloading*), ma sono operazioni diverse.

3.4 Tipo reale (II)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
  int i = 1, j = 2;
   int z = i / j;                          // 0
  float f1 = 1.0 / 2.0;                    // 0.5
  float f2 = 1 / 2;                        // 0
   float f3 = (float)i / j;                // 0.5
  cout << z << '\t' << f1 << '\t' << f2 << '\t' << f3 << endl;
  system("PAUSE");
  return 0;
}
```

```
0       0.5     0       0.5
Premere un tasto per continuare . . .
```

3.5 Tipo bool (I)

Tipo *bool*:

valori: costanti predefinite *false* e *true* (codificati con gli interi 0 e 1, rispettivamente).

Operazioni:

|| OR logico o disgiunzione

&& AND logico o congiunzione

! NOT logico o negazione

p	q	p ||q	p&&q	!p
false	false	false	false	true
false	true	true	false	true
true	false	true	false	false
true	true	true	true	false

3.5 Tipo bool (II)

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    bool b1 = true, b2 = false;
    bool b3 = b1 && b2;                  // b3 = false
    bool b4 = b1 || b2;                  // b4 = true

    bool b5 = b1 || b2 && false;
                                         // b5 = true (AND precedenza
                                         //  maggiore di OR)
    bool b6 = !b2|| b2 && false;
                                         // b6 = true (NOT prec. maggiore di
                                         //  AND e OR)

    cout << b3 << '\t' << b4 << '\t' << b5;
    cout << '\t' << b6 << endl;
    system("PAUSE");
    return 0;
}
```

```
0       1       1       1
Premere un tasto per continuare . . .
```

3.5 Operatori di confronto e logici (I)

I tipi aritmetici possono utilizzare gli operatori di confronto:

- **==** uguale
- **!=** diverso
- **>** maggiore
- **>=** maggiore o uguale
- **<** minore
- **<=** minore o uguale

Operatori di confronto:

- **il risultato è un booleano, che vale *false* se la condizione espressa dall'operatore non è verificata, *true* altrimenti;**
- **gli operatori di confronto si dividono in:**
  - ***operatori di uguaglianza (== e !=);***
  - ***operatori di relazione;***
- **i primi hanno una precedenza più bassa degli altri.**

3.5 Operatori di confronto e logici (II)

```
int main()
{
    bool b1, b2, b3, b4, b5;
    int i = 10;
    float f = 8.0;

    b1 = i > 3 && f < 5.0;      // false
    b2 = i == f < 5.0;          // false
    b3 = i == i;                // true

b4 = 4 < i < 7;             // true ????

    b5 = 4 < i && i < 7;        // false

    cout << b1 << '\t' << b2 << '\t' << b3 << endl;
    cout << b4 << '\t' << b5 << endl;
    return 0;
}
```

```
0       0       1
1       0
Premere un tasto per continuare . . .
```

3.6 Tipo carattere (I)

- **insieme di valori:**
  caratteri opportunamente codificati (generalmente un carattere occupa un byte).

- **operazioni sui caratteri:**
  sono possibili tutte le operazioni definite sugli interi, che agiscono sulle loro codifiche.

Codifica usata:

- dipende dall'implementazione;
- la più comune è quella ASCII.

Letterale carattere:

- carattere racchiuso fra apici;
- esempio:
  - *Il letterale 'a' rappresenta il carattere* a.

3.6 Tipo carattere (I)

Caratteri di controllo:

- rappresentati da combinazioni speciali che iniziano con una barra invertita (sequenze di *escape*).

Alcuni esempi:

- **nuova riga (LF) \n**
- **tabulazione orizzontale \t**
- **ritorno carrello (CR) \r**
- **barra invertita **
- **apice \'**
- **virgolette \"**

3.6 Tipo carattere (II)

Ordinamento:

- tutte le codifiche rispettano l'ordine alfabetico fra le lettere, e l'ordine numerico fra le cifre;
- la relazione fra lettere maiuscole e lettere minuscole, o fra caratteri non alfabetici, non è prestabilita (per esempio, in ASCII si ha 'A' < 'a').

Carattere:

- può essere scritto usando il suo valore nella codifica adottata dall'implementazione (per esempio ASCII). Il valore puo' essere espresso in decimale, ottale ed esadecimale.

3.6 Tipo carattere (II)

Valori ottali:

- formati da cifre ottali precedute da una barra invertita.

Valori esadecimali:

- formati da cifre esadecimali precedute da una barra invertita e dal carattere x (non X).

Nota:

- le sequenze di escape e le rappresentazioni ottale e esadecimale di un carattere, quando rappresentano un *letterale carattere,* vanno racchiuse fra apici;
- esempi:
- '\n' '\15'

3.6 Tipo carattere (III)

```
int main()
{
    char c1 = 'c', t = '\t', d = '\n';
    char c2 = '\x63';              // 'c' (in esadecimale)
    char c3 = '\143';              // 'c' (in ottale)
    char c4 = 99;                  // 'c' (in decimale)
    cout << c1 << t << c2 << t << c3 << t << c4 << d;

    char c5 = c1 + 1;              // 'd'
    char c6 = c1 - 2;              // 'a'
    char c7 = 4 * d + 3;           // '+' (!!!)
    int i = c1 - 'a';              // 2
    cout << c5 << t << c6 << t << c7 <<  t << i << d;

    bool m = 'a' < 'b', n = 'a' > 'c';     // m = true, n = false
    cout << m << t << n << '\n';
    return 0;
}
```

```
c       c       c       c
d       a       +       2
1       0
Premere un tasto per continuare . . .
```

3.7 Tipi enumerazione (I)

Tipi enumerazione (o enumerati):

- costituiti da insiemi di costanti intere, definite dal programmatore, ciascuna individuata da un identificatore e detta *enumeratore*;
- utilizzati per variabili che assumono solo un numero limitato di valori;
- servono a rappresentare informazioni non numeriche;
- non sono predefiniti, ma definiti dal programmatore.

Nota:

- è possibile effettuare separatamente la dichiarazione di un tipo enumerazione e la definizione di variabili di quel tipo.

Operazioni:

- tipicamente, quelle di confronto;
- sono possibili tutte le operazioni definite sugli interi, che agiscono sulla codifica degli enumeratori.

3.7 Tipi enumerazione (II)

```
int main()
{
    enum Giorni {LUN,MAR,MER,GIO,VEN,SAB,DOM};
    Giorni oggi = MAR;
    oggi = MER;

    int i = oggi;           // 2, conversione implicita
//  oggi = MER-MAR;         // ERRORE! MER-MAR->intero
//  oggi = 3;               // ERRORE! 3 costante intera
//  oggi = i;               // ERRORE! i e' un intero

    cout << int(oggi) << endl;      // 2
    cout << oggi << endl;           // 2, conv. implicita

    enum {ROSSO, GIALLO, VERDE} semaforo;
    semaforo = GIALLO;
    cout << semaforo << endl;       // 1
```

3.7 Tipi enumerazione (II)

```
    enum {INIZ1=10, INIZ2, INIZ3=9, INIZ4};
    cout << INIZ1 << '\t' << INIZ2 << '\t';
    cout << INIZ3 << '\t' << INIZ4 << endl;

    return 0;
}
```

```
2
2
1
10      11      9       10
Premere un tasto per continuare . . .
```

3.8.1 Conversioni implicite (I)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    int i = 10, j;
    float f = 2.5, h;
    double d = 1.2e+1;
    char c = 'd';

    h = f + 1;                  // 3.5
    cout << h << '\t';

    j = f + 3.1;                // 5
    cout << j << endl;

    d = i + 1;                  // 11
    cout << d << '\t';
```

3.8.1 Conversioni implicite (I)

```
    d = f + d;                  // 13.5
    cout << d << endl;

    j = c - 'a';                // 3
    cout << j << endl;

    return 0;
}
```

```
3.5     5
11      13.5
3
Premere un tasto per continuare . . .
```

3.8.1 Conversioni implicite (II)

Osservazione:

- nella conversione da *double* a *int* si può avere una perdita di informazione, poiché avviene un troncamento della parte decimale;

- in alcuni casi, nella conversione da *int* a *double* si può verificare una perdita di precisione per arrotondamento, poiché gli interi sono rappresentati in forma esatta ed i reali sono rappresentati in forma approssimata.

- **Esempi:**
  - **il reale 1588.5 convertito nell'intero 1588;**
  - **l'intero 0X7FFFFFF0 (2147483632)**

    convertito nel reale 0X80000000 (2147483648)

3.8.1 Conversioni implicite (II)

Conversioni più significative per gli operatori binari (aritmetici):

- se un operatore ha entrambi gli operandi interi o reali, ma di lunghezza diversa, quello di lunghezza minore viene convertito al tipo di quello di lunghezza maggiore;

- se un operatore ha un operando intero ed uno reale, il valore dell'operando intero viene convertito nella rappresentazione reale, ed il risultato dell'operazione è un reale.

3.8.1 Conversioni implicite (III)

Conversioni più significative per l'assegnamento:

- a una variabile di tipo reale può essere assegnato un valore di tipo intero;
- a una variabile di tipo intero può essere assegnato un valore di tipo reale, di tipo booleano, di tipo carattere o di un tipo enumerazione;
- a una variabile di tipo carattere può essere assegnato un valore di tipo intero, di tipo booleano, o di un tipo enumerazione.

Nota:

- a una variabile di tipo booleano o di un tipo enumerazione non può essere assegnato un valore che non sia del suo tipo.

Conversioni implicite in sequenza:

esempio: a una variabile di tipo reale può essere assegnato un valore di tipo carattere (conversione da carattere a intero, quindi da intero a reale).

3.8.2 Conversioni esplicite

Operatore *static_cast:*

- effettua una conversione di tipo quando esiste la conversione implicita inversa;
- puo essere usato per effettuare conversioni di tipo previste dalla conversione implicita.

3.8.2 Conversioni esplicite

```
int main()
{
    enum Giorni {LUN,MAR,MER,GIO,VEN,SAB,DOM};
    int i; Giorni g1 = MAR, g2, g3;
    i = g1;
    g1 = static_cast<Giorni>(i);
    g2 = (Giorni) i;                    // cast
    g3 = Giorni (i);                    // notazione funzionale

    cout << g1 << '\t' << g2 << '\t' << g3 << endl;

    int j = (int) 1.1;                  // cast, 1
    float f = float(2);                 // notazione funzionale
    cout << j << '\t' << f << endl;

    return 0;
}
```

```
1       1       1
1       2
Premere un tasto per continuare . . .
```

3.9 Dichiarazioni di oggetti costanti

Oggetto costante:

- si usa la parola ***const*** nella sua definizione;
- è richiesto sempre un inizializzatore.

```
int main()
{
    const long int i = 0;
    const double e1 = 3.5;
    const long double e2 = 2L * e1;

    cout << i << '\t' << e1 << '\t' << e2 << endl;

//  i = 3;                  // ERRORE!

//  const int j;            // ERRORE!

    return 0;
}
```

```
0       3.5     7
Premere un tasto per continuare . . .
```

3.10 Operatore sizeof (I)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    cout << "char \t" << sizeof(char) << endl;          // 1
    cout << "short \t" << sizeof(short) << endl;        // 2
    cout << "int \t" << sizeof(int) << endl;            // 4
    cout << "long \t" << sizeof(long) << endl;          // 4

    cout << "unsigned char \t";
    cout << sizeof(unsigned char) << endl;             // 1
    cout << "unsigned short \t";
    cout << sizeof(unsigned short) << endl;            // 2
    cout << "unsigned int \t";
    cout << sizeof(unsigned int) << endl;              // 4
    cout << "unsigned long \t";
    cout << sizeof(unsigned long) << endl;             // 4
```

3.10 Operatore sizeof (I)

```
    cout << "float \t" << sizeof(float) << endl;          // 4
    cout << "double \t" ;
    cout << sizeof(double) << endl;                       // 8
    cout << "long double \t";
    cout << sizeof(long double) << endl;                  // 12
    return 0;
}
```

```
char    1
short   2
int     4
long    4
unsigned char    1
unsigned short   2
unsigned int   4
unsigned long    4
float    4
double  8
long double    12
Premere un tasto per continuare . . .
```

3.10 Operatore sizeof (III)

```
int main()
{
    cout << "costante carattere ";
    cout << sizeof 'c' << endl;                    // 1
    cout << "costante carattere ";
    cout << sizeof('c') << endl;                   // 1

    char c = 0;
    cout << "variabile carattere " << sizeof c << endl;  // 1

//  cout << "char " << sizeof char << endl;  ERRORE!
    return 0;
}
```

```
costante carattere 1
costante carattere 1
variabile carattere 1
Premere un tasto per continuare . . .
```

4.1 Struttura di un programma

basic-main-program

int main () ***compound-statement***

compound-statement

{ ***statement-seq*** }

statement

declaration-statement

definition-statement

expression-statement

structured-statement

jump-statement

labeled-statement

4.1 Struttura di un programma

Istruzioni di dichiarazione/definizione:

declaration-statement

definition-statement

hanno la forma vista in precedenza.

Simboli introdotti dal programmatore:

- devono essere dichiarati/definiti prima di essere usati;
- non è necessario che le dichiarazioni/definizioni precedano le altre istruzioni.

4.2 Espressioni di assegnamento (I)

Sintassi:

expression-statement

expression|opt ;

Espressione:

- formata da letterali, identificatori, operatori, ecc., e serve a calcolare un valore;
- opzionale, perché in certi casi può essere utile usare una istruzione vuota (che non compie alcuna operazione) indicando il solo carattere ';' .

Espressione comune:

- assegnamento (nuovo valore a una variabile);
- sintassi:

basic-assignment-expression

variable-name = expression

4.2 Espressioni di assegnamento (I)

variable-name = expression

Effetto:

- calcolare il valore dell'espressione a destra dell'operatore di assegnamento ('=');
- sostituirlo al valore della variabile.

Nome a sinistra dell'operatore di assegnamento:

- individua una variabile, ossia un *lvalue* (left value).

Espressione a destra dell'operatore di assegnamento :

- rappresenta un valore, ossia un *rvalue* (right value).

4.2 Espressioni di assegnamento (II)

```
int main()
{
    int i = 0, j = 1, k;

    i = 3;
    cout << i << endl;              // 3

    j = i;
    cout << j << endl;             // 3

    k = j = i = 5;                  // associativo a destra
    cout << i << '\t' << j << '\t' << k << endl;                  // 5 5 5

    k = j = 2 * (i = 3);
    cout << i << '\t' << j << '\t' << k << endl;                  // 3 6 6

   //    k = j +1 = 2 * (i = 100);   // ERRORE!

    (j = i) = 10;                   // restituisce un l-value
    cout << j << endl;             // 10

 return 0;
}
```

4.2.1 Altri operatori di assegnamento

basic-recurrence-assignment
variable-name = variable-name op ***expression***

basic-compound-assignment
variable-name op ***= expression***

```
int main()
{
    int i = 0, j = 5;
    i += 5;                    // i = i + 5
    cout << i << endl;         // 5

    i *= j + 1;                // i = i * (j + 1);
    cout << i << endl;         // 30

    i - = j - = 1;             // associativo a destra;
    cout << i << endl;         // 26

    (i += 12) = 2;             // restituisce un l-value
    cout << i << endl;         // 2
    return 0;
}
```

4.2.2 Incremento e decremento

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
     int i, j;

    i = 0; j = 0;
    ++i; --j;                                  // i += 1; j -= 1;
    cout << i << '\t' << j << endl;        // 1 -1

    i = 0;
    j = ++i;                                       // i += 1; j = i;
    cout << i << '\t' << j << endl;        // 1 1

    i = 0;
    i++;
    cout << i << endl;                         // 1
```

4.2.2 Incremento e decremento

```
    i = 0;
    j = i++;                                    // j = i; i += 1;
    cout << i << '\t' << j << endl;             // 1  0

   // j = ++i++;                                // ERRORE!
      j = (++i)++;

  // j = i++++;                                 // ERRORE!
    int k = ++++i;

    cout << i << '\t' << j << '\t' << k << endl;                    // 5  2  5

    return 0;
}
```

4.3 Espressioni aritmetiche e logiche (I)

Calcolo delle espressioni:

- vengono rispettate le precedenze e le associatività degli operatori

Precedenza:

-per primi vengono valutati i fattori, calcolando i valori delle funzioni e applicando gli operatori unari (prima incremento e decremento postfissi, poi incremento e decremento prefissi, NOT logico (!), meno unario (-) e più unario (+));

- poi vengono valutati i termini, applicando gli operatori binari nel seguente ordine:
  - quelli moltiplicativi (*, / , %);
  - quelli additivi (+ , -);
  - quelli di relazione (<, …);
  - quelli di uguaglianza (==, !=)
  - quelli logici (nell'ordine, &&, ||);
  - quelli di assegnamento (= , …);

Parentesi tonde (coppia di separatori):

- fanno diventare qualunque espressione un fattore, che viene quindi calcolato per primo.

4.3 Espressioni aritmetiche e logiche (II)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    int i = 2, j;
    j = 3 * i + 1;
    cout << j << endl;          // 7

    j = 3 * (i + 1);
    cout << j << endl;          // 9


    return 0;
}
```

4.3 Espressioni aritmetiche e logiche (III)

Associatività:

- gli operatori aritmetici binari sono associativi a sinistra;
- gli operatori unari sono associativi a destra;
- gli operatori di assegnamento sono associativi a destra.

```
int main()
{
    int i = 8, j = 4, z;

    z = i / j / 2;
    cout << z << endl;          // 1

    z = i / j * 2;
    cout << z << endl;          // 4

    z = i / ( j * 2 );
    cout << z << endl;          // 1

    z = j * 2 / i;
    cout << z << endl;          // 1
    return 0;
}
```

4.3 Espressioni aritmetiche e logiche (IV)

```
int main()
{
    bool k;
    int i = 0, j = 5;

    k = i >= 0 && j <= 1;          // (i >= 0) && (j <= 1)
    cout << k << endl;             // 0

    k = i && j || !k;              // (i && j) || (!k)
    cout << k << endl;             // 1

    k = 0 < j < 4;                 // ATTENZIONE!
    cout << k << endl;             // 1

    k = 0 < j && j < 4;
    cout << k << endl;             // 0

    return 0;
}
```

4.3 Espressioni aritmetiche e logiche (V)

Operatori && e ||:

- **sono associativi a sinistra;**
- **il calcolo di un'espressione logica contenente questi operatori termina appena si può decidere se l'espressione è, rispettivamente, falsa o vera.**

```
                    // Cortocircuito

int main()
{
    bool k;
    int i = 0;

    k = (i >= 0) || (i++);
    cout << k << '\t' << i << endl;          // 1 0
```

4.3 Espressioni aritmetiche e logiche (V)

```
    k = (i > 0) || (i++);
    cout << k << '\t' << i << endl;          // 0 1

    k = (i >= 0) && (i <= 100);
    cout << k << endl;                       // 1

    k = (i != 0) && (10 / i >= 10);          // OK
    cout << k << endl;

    k = (10 / i >= 10) && ( i != 0);         // NO!
    cout << k << endl;


    return 0;
}
```

4.4 Operatore condizionale (I)

e1* ? *e2* : *e3

***e1* espressione logica**

Se *e1* è vera, il valore restituito dall'operatore condizionale è il valore di *e2,* altrimenti di *e3.*

```
// Legge due interi e stampa su uscita standard il minore
int main()
{
    int i, j, min;
    cout << "Inserisci due numeri interi" << endl;

    cin >> i >> j;                                  // 2  4
    min = (i < j ? i : j);
    cout << "Il numero minore e': " << min << endl;
    return 0;
}
```

```
Inserisci due numeri interi
2
4
Il numero minore e': 2
```

4.4 Operatore condizionale (II)

```
// Legge tre interi ed incrementa il minore

int main()
{
    int i, j, z, min;
    cout << "Inserisci tre numeri interi" << endl;
    cin >> i >> j >> z;                                    // 2 3 4

    min = i < j ? (i<z ? i++ : z++) : (j < z ? j++ : z++);
    cout << "Il numero minore e': " << min << endl;
    cout << i << '\t' << j << '\t' << z << endl;

    return 0;
}
```

```
Inserisci tre numeri interi
2
3
4
Il numero minore e': 2
3       3       4
```

6. Istruzioni strutturate

Istruzioni strutturate:

- **consentono di specificare azioni complesse.**

structured-statement

compound-statement

selection-statement

iteration-statement

Istruzione composta:

- già esaminata nella sintassi di programma;
- consente, per mezzo della coppia di delimitatori { e } , di trasformare una qualunque sequenza di istruzioni in una singola istruzione.
- ovunque la sintassi preveda un'istruzione, si può mettere una sequenza comunque complessa di istruzioni racchiusa tra i due delimitatori.

6. Istruzioni strutturate

Istruzioni condizionali:

selection-statement

if-statement

switch-statement

---

if-statement

if (***condition***) ***statement***

if (***condition***) ***statement*** else ***statement***

6.3.1 Istruzione if (I)

```
// Trova il maggiore tra due interi

#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    int a, b, max;
    cin >> a >> b;                    // 4 6
    if (a > b)
        max = a;
    else
        max = b;
    cout << max << endl;

    return 0;
}
```

```
4
6
6
```

6.3.1 Istruzione if (II)

```
// Incrementa o decrementa

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    cin >> a >> b;
    if (a >= b)
    {
        a++;
        b++;
    }
    else
    {
        a--;
        b--;
    }
    cout << a << '\t' << b << endl;
    return 0;
}
```

```
4
6
3       5
```

6.3.1 Istruzione if (III)

```
// Valore assoluto (if senza parte else)

#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    int a;
    cout << "Inserisci un numero intero " << endl;
    cin >> a;
    if (a < 0)
        a = -a;
    cout << "Il valore assoluto e' ";
    cout << a << endl;

    return 0;
}
```

```
Inserisci un numero intero
-4
Il valore assoluto e' 4
```

6.3.1 Istruzione if (IV)

```
// Legge un numero, incrementa il numero e
// lo scrive se è diverso da 0
// (if senza espressione relazionale)

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "Inserisci un numero intero " << endl;
    cin >> i;
    if (i++)
        cout << "Numero incrementato " << i << endl;

        return 0;
}
```

```
Inserisci un numero intero
2
Numero incrementato 3
```

```
Inserisci un numero intero
0
```

N.B.: L'espressione nella condizione può restituire un valore aritmetico: se il valore è 0, la condizione è falsa; altrimenti è vera.

6.3.1 Istruzione if (V)

```
// If in cascata
// if (a > 0 ) if ( b > 0 ) a++; else b++;

int main()
{
    int a, b;
    cout << "Inserisci due numeri interi" << endl;
    cin >> a >> b;
    if (a > 0)
        if (b > 0)
            a++;
        else
            b++;
    cout << a << '\t' << b << endl;

    return 0;
}
```

NOTA BENE
la parte *else* si riferisce alla condizione più vicina (nell'esempio, alla condizione b > 0);

```
Inserisci due numeri interi
3
5
4       5
```

6.3.1 Istruzione if (VI)

```
// Scrittura fuorviante

int main()
{
    int a, b;
    cout << "Inserisci due numeri interi" << endl;
    cin >> a >> b;
    if (a > 0)
        if (b > 0)
            a++;
    else
        b++;
    cout << a << '\t' << b << endl;

    return 0;
}
```

```
Inserisci due numeri interi
5
7
6       7
```

6.3.1 Istruzioni if (VII)

```
// Scrive asterischi

int main()
{
    int i;
    cout << "Quanti asterischi? " << endl;
    cin >> i;
    if (i == 1)
        cout << '*';
    else
        if (i == 2)
            cout << "**";
        else
            if (i == 3)
                cout << "***";
            else
                cout << '!';
    cout << endl;

    return 0;
}
```

```
Quanti asterischi?
2
**
```

6.3.1 Istruzione if (VIII)

```
// Equazione di secondo grado

#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    double a, b, c;
    cout << "Coefficienti? " << endl;
    cin >> a >> b >> c;
```

6.3.1 Istruzione if (VIII)

```
    if ((a == 0) && (b == 0))
        cout << "Equazione degenere" << endl;
    else
        if (a == 0)
        {
            cout << "Equazione di primo grado" << endl;
            cout << "x: " << -c / b << endl;
        }
        else
        {
            double delta = b * b - 4 * a * c;
            if (delta < 0)
                cout << "Soluzioni immaginarie" << endl;
            else
            {
                delta = sqrt(delta);
                cout << "x1: " << (-b + delta) / (2 * a) << endl;
                cout << "x2: " << (-b - delta) / (2 * a) << endl;
            }
        }
    return 0;
}
```

```
Coefficienti?
1
6
9
x1: -3
x2: -3
```

6.3.2 Istruzioni switch e break (I)

Sintassi:

switch-statement

switch (***condition***) ***switch-body***

switch-body

{ ***alternative-seq*** }

alternative

case-label-seq ***statement-seq***

case-label

case ***constant-expression*** :

default :

6.3.2 Istruzioni switch e break (I)

Condizione:

- comunemente costituita da un'espressione, che produce un risultato di tipo discreto;

Etichette (*case-label*):

- contengono (oltre alla parola chiave *case*) espressioni costanti il cui valore deve essere del tipo del risultato dell'espressione;
- individuano le varie alternative nel corpo dell'istruzione *switch*;
- i valori delle espressioni costanti devono essere distinti.

Alternativa con etichetta *default*:

- se presente, deve essere unica.

Terminazione:

può essere ottenuta con l'istruzione break (categoria delle istruzioni di salto):

break-statement

break ;

6.3.2 Istruzioni switch e break (II)

Esecuzione dell'istruzione *switch*:

- viene valutata l'espressione;
- viene eseguita l'alternativa con l'etichetta in cui compare il valore calcolato (ogni alternativa può essere individuata da più etichette);
- se nessuna alternativa ha un'etichetta in cui compare il valore calcolato, allora viene eseguita, se esiste, l'alternativa con etichetta *default*;
  - *in mancanza di etichetta* default *l'esecuzione dell'istruzione* switch *termina.*

Alternativa:

- formata da una o più istruzioni (eventualmente vuote o strutturate).

Attenzione:

- Se l'ultima istruzione di un'alternativa non fa terminare l'istruzione *switch*, e se l'alternativa non e' l'ultima, viene eseguita l'alternativa successiva.

6.3.2 Istruzioni switch e break (III)

```
// Scrive asterischi (uso istruzione break)

int main()
{    int i;
     cout << "Quanti asterischi? " << endl;
     cin >> i;
     switch (i)
     {
          case 1:
               cout << '*';
               break;
          case 2:
               cout << "**";
               break;
          case 3:
               cout << "***";
               break;
          default:
               cout << '!';
     }
     cout << endl;
     return 0;
}
```

```
Quanti asterischi?
2
**
```

6.3.2 Istruzioni switch e break (IV)

```
// Scrive asterischi (in cascata)

int main()
{
    int i;
    cout << "Quanti asterischi? " << endl;
    cin >> i;
    switch (i)
    {
        case 3:                          // in cascata
            cout << '*';
        case 2:                          // in cascata
            cout << '*';
        case 1:
            cout << '*';
            break;
        default:
            cout << '!';
    }
    cout << endl;
    return 0;
}
```

```
Quanti asterischi?
2
**
```

6.3.2 Istruzioni switch e break (V)

```
// Creazione menù
// Selezione tramite caratteri

#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    cout << "Seleziona un'alternativa" << endl;
    cout << "A - Prima Alternativa" << endl;
    cout << "B - Seconda Alternativa" << endl;
    cout << "C - Terza Alternativa" << endl;
    cout << "Qualsiasi altro tasto per uscire" << endl;

    char c;
    cin >> c;
```

6.3.2 Istruzioni switch e break (V)

```
    switch (c)
    {
        case 'a': case 'A':
            cout << "Prima alternativa" << endl;
            break;
        case 'b': case 'B':
            cout << "Seconda alternativa" << endl;
            break;
        case 'c': case 'C':
            cout << "Terza alternativa" << endl;

    // Manca il caso di default
    // Se non è una delle alternative, non scrive niente

    }

    return 0;
}
```

6.3.2 Istruzioni switch e break (VI)

```
// Creazione menù
// Selezione tramite caratteri
```

```
Seleziona un'alternativa
A - Prima Alternativa
B - Seconda Alternativa
C - Terza Alternativa
Qualsiasi altro tasto per uscire
B
Seconda alternativa
```

6.3.2 Istruzioni switch e break (VII)

```
// Scrittura di enumerazioni
int main()
{
    enum {ROSSO, GIALLO, VERDE} colore;
    char c;
    cout << "Seleziona un colore " << endl;
    cout << "R - rosso " << endl;
    cout << "G - giallo " << endl;
    cout << "V - verde " << endl;

    cin >> c;
    switch (c)
    {
        case 'r': case 'R':
            colore = ROSSO;
            break;
        case 'g': case 'G':
            colore = GIALLO;
            break;
        case 'v': case 'V':
            colore = VERDE;
    }
```

6.3.2 Istruzioni switch e break (VIII)

```
// Scrittura di enumerazioni (continua)

    switch (colore)
    {
        case ROSSO:
            cout << "ROSSO";
            break;
        case GIALLO:
            cout << "GIALLO";
            break;
        case VERDE:
            cout << "VERDE";
    }
    cout << endl;

    return 0;
}
```

```
Seleziona un colore
R - rosso
G - giallo
V - verde
v
VERDE
```

6.4.1 Istruzione ripetitive

Sintassi:

iteration-statement

while-statement

do-statement

for-statement

---

while-statement

while (***condition***) ***statement***

6.4.1 Istruzione ripetitive

```
// Scrive n asterischi, con n dato (i)

#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    int n, i;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    i = 0;
    while (i < n)
    {
        cout << '*';
        i++;
    }                                   // n conserva il valore iniziale
    cout << endl;

    return 0;
}
```

```
Quanti asterischi?
6
******
```

6.4.1 Istruzione while (I)

```
// Scrive n asterischi, con n dato (ii)

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;

    while (n > 0)
    {
        cout << '*';
        n--;
    }                                   // al termine, n vale 0

    cout << endl;
    return 0;
}
```

```
Quanti asterischi?
6
******
```

6.4.1 Istruzione while (II)

```
// Scrive n asterischi, con n dato (iii)
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    while (n-- > 0)
        cout << '*';                    // al termine, n vale -1
    cout << endl;

    return 0;
}
```

6.4.1 Istruzione while (II)

```
// Scrive n asterischi, con n dato (iv)
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;

    while (n--)                    // non termina se n < 0
        cout <<'*';

    cout << endl;

    return 0;
}
```

6.4.1 Istruzione while (III)

```
// Legge, raddoppia e scrive interi non negativi
// Termina al primo negativo

int main()
{
    int i;
    cout << "Inserisci un numero intero" << endl;
    cin >> i;

    while (i >= 0)
    {
        cout << 2 * i << endl;
        cout << "Inserisci un numero intero" << endl;
        cin >> i;
    }

    return 0;
}
```

```
Inserisci un numero intero
1
2
Inserisci un numero intero
3
6
Inserisci un numero intero
-2
```

6.4.1 Istruzione while (IV)

```
// Calcola il massimo m tale che la somma dei primi
// m interi positivi e` minore o uguale ad un dato intero
// positivo n

int main()
{
    unsigned int somma = 0, m = 0, n;

    cout << "Inserisci n " << endl;
    cin >> n;

    while (somma <= n) {
        m++;
        somma += m;
    }

    m--;
    cout << m << endl;

    return 0;
}
```

```
Inserisci n
8
3
```

6.4.1 Istruzione while (IV)

m 0 somma 0 n 8

somma <= n

0 < = 8	->	m 1	somma 1	
1 < = 8	->	m 2	somma 3	
3 < = 8	->	m 3	somma 6	
6 < = 8	->	m 4	somma 10	
10 < = 8	->	m 3	somma 10	

6.4.1 Istruzione while (V)

Serie di Fibonacci:

$a_0 = 0$
$a_1 = 1$

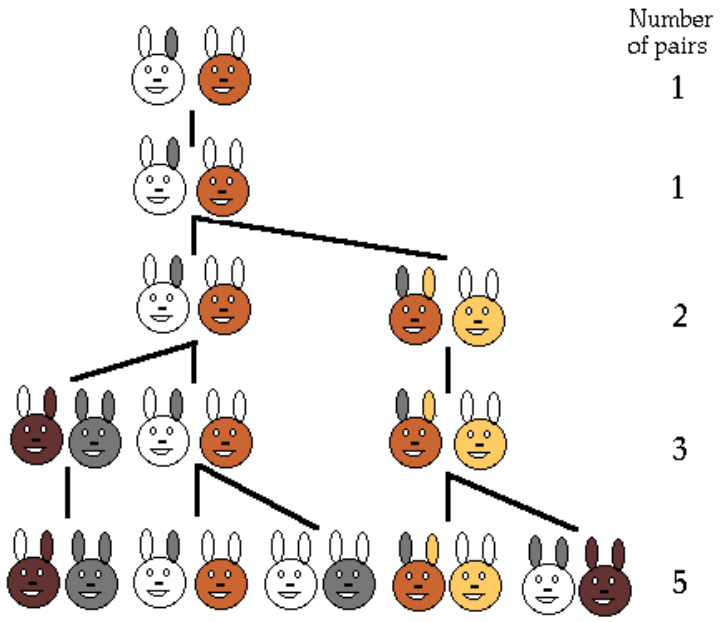
$a_n = a_{n-1} + a_{n-2}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……

6.4.1 Istruzione while (V)

Curiosità: da dove nasce la serie di Fibonacci?

Supponiamo di avere una coppia di conigli (maschio e femmina). I conigli sono in grado di riprodursi all'età di un mese. Supponiamo che i nostri conigli non muoiano mai e che la femmina produca sempre una nuova coppia (un maschio ed una femmina) ogni mese dal secondo mese in poi. Il problema posto da Fibonacci fu: quante coppie ci saranno dopo un anno?



Il numero delle coppie di conigli all'inizio di ciascun mese sarà

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

6.4.1 Istruzione while (V)

```
// Calcola il termine n-esimo della successione di  Fibonacci  con n >= 3

int main()
{
     int i = 0, j = 1, s, n;                                  // i = a0, j = a1, s
     cout << "Inserisci n " << endl;
     cin >> n;
     if (n <= 2 ) cout << "Valore non consistente" << endl;
     else
       { quanti = n-2;
          while (quanti > 0)
             {
                s = i + j;
                i = j;
                j = s;
                quanti--;
             }
          cout << j << endl;
    }
     return 0;
}
```

```
Inserisci n
7
8
```

6.4.1 Istruzione while (V)

```
// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n

int main()
{
    int i = 0, j = 1, s, n;                    // i = a0, j = a1, s
    cout << "Inserisci n " << endl;
    cin >> n;

    if (n<=0) cout << "Valore non consistente" << endl;
    else
    {
        while ((s = j + i) <= n)
        {
            i = j;
            j = s;
         }
        cout << j << endl;
   }

    return 0;
}
```

```
Inserisci n
7
5
```

6.4.1 Istruzione while (V)

```
// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n

//    Soluzione con due variabili

int main()
{
     int i = 0, j = 1,n;                    // i = a0, j = a1
     cout << "Inserisci n " << endl;
     cin >> n;
     if (n<=0) cout << "Valore non consistente" << endl;
     else
     {
          while (((i = j+i)<= n)&&((j = j+i)<=n));

          if (j<i) cout << j << endl;
          else cout << i << endl;
     }
     return 0;
}
```

```
Inserisci n
7
5
```

6.4.2 Istruzione do (I)

do-statement

do ***statement*** while (***expression***) ;

Esecuzione dell'istruzione do:

- viene eseguita l'istruzione racchiusa tra *do* e *while* (corpo del *do*);
- viene valutata l'espressione;
- se questa risulta vera l'istruzione *do* viene ripetuta;
- se questa risulta falsa l'istruzione *do* termina.

Nota:

- il corpo dell'istruzione *do* viene eseguito almeno una volta, prima della valutazione della condizione di terminazione.

6.4.2 Istruzione do (II)

```
// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n

int main()
{
    int i, j = 0, s = 1, n;
    cout << "Inserisci n " << endl;
    cin >> n;
    if (n<=0) cout << "Valore non consistente" << endl;
    else
    {
        do
        {
            i = j;
            j = s;
        } while ((s = j + i) <= n);

         cout << j << endl;
    }

    return 0;
}
```

```
Inserisci n
7
5
```

6.4.3 Istruzione for (I)

for-statement
for (***initialization*** ***condition-specification*** ***step***)
statement

initialization
expression-statement
definition-statement

condition-specification
expression**|opt** ;

step
expression**|opt**

6.4.3 Istruzione for (I)

Istruzione *for*:

– viene eseguita come se fosse scritta nel seguente modo:

```
{ inizializzazione
  while ( espressione condizionale )
   {  // corpo del for
      istruzione
      passo ;
  }
}
```

6.4.3 Istruzione for (II)

```
// Scrive n asterischi, con n dato (i)

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;

    for (int i = 0; i < n; i++)
        cout << '*';                // al termine, i vale n

    cout << endl;

    return 0;
}
```

```
Quanti asterischi?
6
******
```

6.4.3 Istruzione for (III)

```
// Scrive n asterischi, con n dato (ii)

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti asterischi? " << endl;
    cin >> n;
    for (; n > 0; n--)
        cout << '*';              // al termine, n vale 0

    cout << endl;


    return 0;
}
```

```
Quanti asterischi?
6
******
```

6.4.3 Istruzione for (IV)

```
// Scrive asterischi e punti esclamativi (I)

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "Quanti? " << endl;
    cin >> n;

    for (int i = 0; i < n; i++)
        cout << '*';
    cout << endl;

    for (int i = 0; i < n; i++)          // visibilita' i limitata
        cout << '!';                     // blocco for
    cout << endl;

    return 0;
}
```

```
Quanti?
6
******
!!!!!!
```

6.4.3 Istruzione for (V)

```
// Scrive asterischi e punti esclamativi (II)

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int n,i;
    cout << "Quanti? " << endl;
    cin >> n;

    for (i = 0; i < n; i++)
        cout << '*';
    cout << endl;

    for (i = 0; i < n; i++)
            cout << '!';
    cout << endl;


    return 0;
}
```

```
Quanti?
6
******
!!!!!!
```

. . .

6.4.3 Istruzione for (VI)

```
// Scrive una matrice di asterischi formata da r righe
// e c colonne, con r e c dati

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int r, c;
    cout << "Numero di righe? " << endl;
    cin >> r;
    cout << "Numero di colonne? " << endl;
    cin >> c;
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            cout << '*';
        cout << endl;
    }

    return 0;
}
```

```
Numero di righe?
3
Numero di colonne?
4
****
****
****
```

6.5 Istruzioni di salto

jump-statement

break-statement

continue-statement

goto-statement

return-statement

- **Istruzione break (già vista):**
  - salto all'istruzione immediatamente successiva al corpo del ciclo o dell'istruzione switch che contengono l'istruzione *break*:

```
while ( ... )
{   ...
    break;
    ...
}
```

```
switch ( .... )
{   ....
    break;
    ...
}
```

6.5.1 Istruzione break (I)

```
// Legge e scrive interi non negativi
// Termina al primo negativo

int main()
{
    int j;
    for (;;)      // ciclo infinito; altra forma: while (1)
    {
        cout << "Inserisci un numero intero " << endl;
        cin >> j;
        if (j < 0)
            break;
        cout << j << endl;
    }

    return 0;
}
```

```
Inserisci un numero intero
3
3
Inserisci un numero intero
5
5
Inserisci un numero intero
-1
```

6.5.1 Istruzione break (II)

```
// Legge e scrive al piu` cinque interi non negativi
// Termina al primo negativo

int main()
{
    const int N = 5;
    for (int i = 0, j; i < N; i++)
    {
        cout << "Inserisci un numero intero " << endl;
        cin >> j;

        if (j < 0)
            break;

        cout << j << endl;
    }

        return 0;
}
```

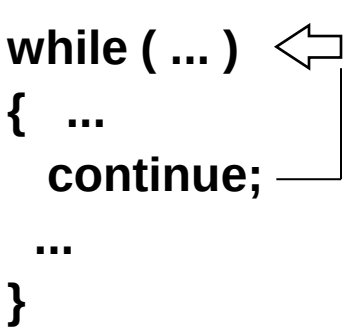
```
Inserisci un numero intero
3
3
Inserisci un numero intero
5
5
Inserisci un numero intero
-1
```

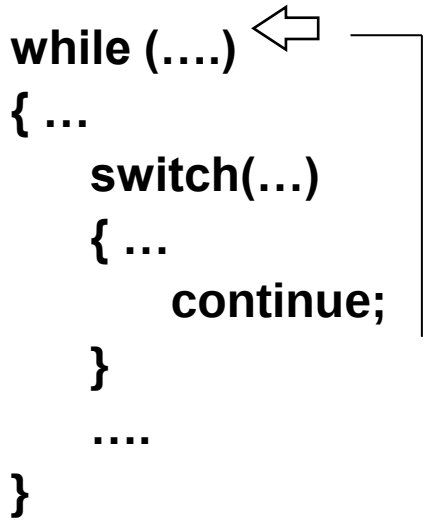
6.5.2 Istruzione continue (I)

continue-statement

continue ;

– provoca la terminazione di un'iterazione del ciclo che la contiene;
– salta alla parte del ciclo che valuta di nuovo la condizione di controllo:





6.5.2 Istruzione continue (II)

```
// Legge cinque interi e scrive i soli non
      negativi

#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
      const int N = 5;
      for (int i = 0, j; i < N; i++)
      {
            cout << "Inserisci un numero intero ";
            cout << endl;

            cin >> j;
            if (j < 0) continue;
            cout << j << endl;
      }

      return 0;
}
```

```
Inserisci un numero intero
1
1
Inserisci un numero intero
2
2
Inserisci un numero intero
-2
Inserisci un numero intero
-3
Inserisci un numero intero
4
4
```

6.5.2 Istruzione continue (|II)

Nota:

- le istruzioni break e continue si comportano in modo diverso rispetto al tipo di istruzione strutturata in cui agiscono;

- l'istruzione continue “ignora” la presenza di un eventuale istruzione switch.

Istruzione switch:

- quando è l'ultima di un ciclo, nelle alternative si può usare l'istruzione continue invece che l'istruzione break.